OULUN AMMATTIKORKEAKOULU

Kimmo Lepikkö

SIMULAATTORI OSANA ARM 6LOWPAN -OHJELMISTOPINON TESTAUSTA

SIMULAATTORI OSANA ARM 6LOWPAN -OHJELMISTOPINON TESTAUSTA

Kimmo Lepikkö
Opinnäytetyö
Kevät 2016
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, langattomat laitteet

Tekijä: Kimmo Lepikkö
Opinnäytetyön nimi: Simulaattori osana ARM 6LoWPAN -ohjelmistopinon testausta
Työn ohjaaja: Terhi Holappa
Työn valmistumislukukausi- ja vuosi: Kevät 2016 Sivumäärä: 33

Opinnäytetyön tarkoituksena oli kehittää ARM 6LoWPAN -ohjelmistopinon testausta hyödyntäen yrityksen sisäisesti kehittämää simulaattoria, jonka avulla voidaan mallintaa suuren sensoriverkon toimintaa ohjelmallisesti. Opinnäytetyön tilaajana toimi ARM Finland Oy. Opinnäytetyön alussa luotiin vaatimusmäärittely simulaattorin jatkokehityksestä, joka mahdollistaisi sen integroinnin yhteen muun testausjärjestelmän kanssa.

Vaatimusmäärittelyn perusteella aloitettiin kehitystyö simulaattorin ja muun testausjärjestelmän kanssa. Aluksi simulaattori piti saada toimimaan yhdessä Python-pohjaisen testauskehyksen kanssa, jotta ohjelmat voisivat vaihtaa viestejä keskenään. Seuraavaksi simulaattoria käyttävään sovellukseen piti lisätä tuki 6LoWPAN-reunareitittimelle sekä reitittävälle noodille. Tämän jälkeen sovellukseen lisättiin tuki liikenteen salaukselle sekä erinäisiä konfiguroituvia parametreja, joilla voidaan muokata simuloitujen noodien asetuksia käynnistyksen yhteydessä. Näiden toimien jälkeen voitiin luoda testauskehyksen avulla testiskriptejä, joilla voitiin testata ohjelmistopinon eri ominaisuuksia automaattisesti.

Opinnäytetyön lopputuloksena 6LoWPAN-ohjelmistopinoa käyttävä simulaattori saatiin yhdistettyä muuhun testausjärjestelmään, jonka avulla voitiin suorittaa simuloituja testejä aina, kun ohjelmistopinoon tulee muutoksia ja mahdolliset ohjelmointivirheet löydetään mahdollisimman aikaisessa vaiheessa. Simulaattoriin pohjautuva testausjärjestelmä on otettu yrityksessä käyttöön pääasialliseksi ja päivittäiseksi testaustyökaluksi ja 6LoWPAN-ohjelmistopinolle on kirjoitettu lyhyessä ajassa useita satoja automaattisia testejä.

Simulaattoriin pohjautuva testausjärjestelmä on jatkuvan kehitystyön alla ja tavoitteena on suurentaa simuloitavan verkon kokoa sekä kasvattaa 6LoWPAN-ohjelmistopinon testattavien ominaisuuksien määrää mahdollisimman monipuoliseksi.

Asiasanat: ohjelmistotestaus, simulaatio, testausautomaatio, 6LoWPAN, sensoriverkko

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Wireless Devices

Author: Kimmo Lepikkö
Title of thesis: Simulator as a part of ARM 6LoWPAN software stack testing
Supervisor: Terhi Holappa
Term and year when the thesis was submitted: Spring 2016 Number of pages: 33

The purpose of the thesis was to develop the ARM 6LoWPAN software stack testing by using the in-house developed simulator as a test tool. The simulator is able to model a large sensor network behavior with software. The thesis was done for ARM Finland Ltd.

The first task for the thesis was to collect all the future requirements for the simulator development in order to integrate it to be used as a part of the test infrastructure. Based on the collected requirements the first development task was to connect the simulator to the Python based test framework so that the two software components could exchange messages between each other. The next task was to add 6LoWPAN border router and router support for the test application that is using the simulator.

After the initial support was ready for the simulated 6LoWPAN devices, support for different security modes and configurable parameters were added to the test application. After that it was possible to create test scripts with the test framework in order to test various features of the 6LoWPAN software stack.

The simulator based test system proved to be very useful for the company's needs and is now the primary software testing tool being used. Simulated test cases are triggered automatically every time there is a new commit to the 6LoWPAN software stack and possible bugs are found early on. Several hundred automated test cases have been written for the software stack in a short time by the test engineers.

The simulator based test system is constantly under development and future goals are to increase the size of the simulated mesh network size and to make the feature testing as versatile as possible.

Keywords: software testing, simulation, test automation, 6LoWPAN, wireless sensor network

ALKULAUSE

Opinnäytetyö tehtiin ARM Finland Oy:lle kesän 2015 aikana ja kirjoitusprosessi vuoden 2016 alussa. Haluan erityisesti kiittää yrityksen johtoa heidän joustavuudestaan ja avustaan, jonka ansiosta sain suoritettua kaikki tutkintoon vaadittavat projektityöt, pakolliset harjoittelut sekä opinnäytetyön samassa yrityksessä. Haluan myös kiittää ryhmääni kuuluneita insinöörejä, jotka väsymättä jaksoivat auttaa opinnäytetyössäni vastaan tulleissa ongelmissa.

Oulussa 13.5.2016

Kimmo Lepikkö

SISÄLLYS

SANASTO

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks, IPv6-osoitteita tukeva, vähävirtaisille laitteille tarkoitettu sensoriverkko
DODAG	Destination Oriented Directed Acyclic Graph, RPL-protokollan käyttämä suuntavektorimalli pakettien reititystä varten
DUT	Device Under Test, testattavana oleva laite
IEEE 802.15.4	Vähävirtaisten laitteiden fyysisen ja MAC-kerroksen standardi
IPv6	128-bittisiä verkko-osoitteita tukeva, IPv4:n korvaava protokolla
MAC	TCP/IP-protokollan siirtoyhteyskerros
Mesh	Verkkomainen rakenne, jonka solmukohdissa on reitittävä laite
ND	Neighbor Discovery, laitteiden naapurinmuodostusprotokolla
Noodi	Node, yleisnimitys sensoriverkon laitteelle
PANA	Protocol for Carrying Authentication for Network Access, luotettava kuljetuskerroksen laitteiden välinen todennusmenetelmä
PSK	Pre-shared Key, kuljetuskerroksen salausmenetelmä
RPL	Routing Protocol for Low-Power and Lossy Networks, pakettien reitityspro-tokolla vähävirtaisille, langattomille laitteille
TUN	TUNnel, simuloi verkkokerroksella IP-paketteja välittävää verkkolaitetta
Valgrind	Työkalu, joka etsii virheitä suoritettavasta ohjelmasta
Wireshark	Ohjelma verkossa liikkuvien pakettien analysointia varten

1 JOHDANTO

Sensoriverkot koostuvat pienistä sulautetuista tietokoneista, jotka ovat langattomasti yhteydessä toisiinsa. Sensoreiden avulla mitataan ympäristön ominaisuuksia, jonka perusteella voidaan suorittaa automaattisesti eri toimintoja, kuten katulamppujen älykästä ohjausta valoisuuden perusteella. Sensorit ovat hyvin vähävirtaisia ja vaativat ohjelmistolta erityisratkaisuja toimiakseen pitkään ja luotettavasti. Lähitulevaisuudessa esineiden internetin odotetaan kasvavan suureksi markkinaksi, jonka tavoitteena on yhdistää kaikki nämä lukemattomat sensorit internetiin.

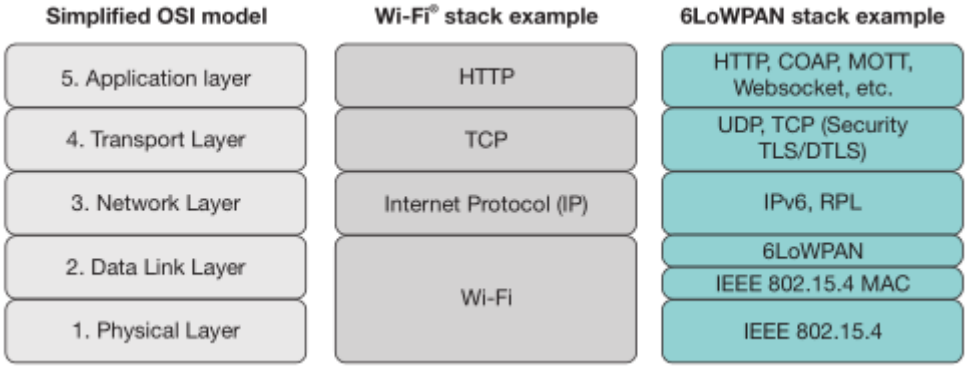
Opinnäytetyön tilaajana toimi ARM Finland Oy, joka on ARM Holdingsin tytäryhtiö ja työllistää Oulussa tällä hetkellä yli 60 henkilöä. Yritys kehittää esineiden internetin palveluja yrityksille sulautettujen laitteiden ohjelmistopinosta lähtien aina pilvipalveluihin saakka. Yrityksessä otettiin juuri käyttöön sensoreille tarkoitettu uusi versio 6LoWPAN-ohjelmistopinosta, jonka testausta oli tarkoitus kehittää opinnäytetyössä simulaattorin avulla.

Jotta simulaattoria voitaisiin hyödyntää testauksessa, tuli se yhdistää osaksi muuta testausjärjestelmää. Opinnäytetyö aloitettiin vaatimusmäärittelyn luomisella simulaattorin jatkokehityksestä, jonka perusteella aloitettiin itse kehitystyö. Ensimmäisessä vaiheessa piti luoda yhteys simulaattorin ja Python-pohjaisen testauskehyksen välille, jotta ohjelmat voisivat vaihtaa viestejä ja komentoja keskenään. Toisessa vaiheessa simulaattoria käyttävää testaussovellusta piti laajentaa tukemaan 6LoWPAN-reunareititintä ja reitittävää noodia, mahdollisuus käyttää yleisimpiä salausmenetelmiä sekä komentoparametrien välittäminen laitteille käynnistyksen yhteydessä. Viimeisessä vaiheessa tuli kirjoittaa ohjelmistopinolle testauskehyksen avulla ensimmäiset automaattiset testit, joita oli mahdollista suorittaa jatkuvassa käännös- ja integrointiympäristössä.

2 6LOWPAN-SOVITUSKERROS

6LoWPANin pohjana toimii IEEE 802.15.4 -standardi, joka määrittelee fyysisen ja MAC-kerroksen vähävirtaisille ja langattomille sulautetuille laitteille, jotka toimivat 2,4 GHz:n, 915 MHz:n ja 868 MHz:n taajuudella (1, s. 18). Tähän standardiin pohjautuvat laitteet eivät ole kuitenkaan suoraan yhteensopivia IP-protokollan kanssa, joten niihin ei saa suoraan yhteyttä muualta internetistä ilman, että välissä olevaan yhdyskäytävään on lisätty tätä varten erillinen tuki. Tämän ongelman ratkaisemiseksi IETF-työryhmä alkoi kehittää 6LoWPAN-standardia. (2, s. 8.)

6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) koostuu joukosta standardeja, joiden avulla voidaan tehokkaasti käyttää IPv6-osoitteita hyväksi vähävirtaisilla, alhaisen tiedonsiirtonopeuden omaavilla langattomilla sulautetuilla laitteilla (1, s. 6, 195). 6LoWPAN-protokollapinoon on lisätty sovituskerros siirtokerroksen ja verkkokerroksen väliin, jonka avulla IPv6-paketteja voidaan välittää IEEE 802.15.4 -standardiin pohjautuvilla radiolähettimillä (kuva 1) (3, s. 4).

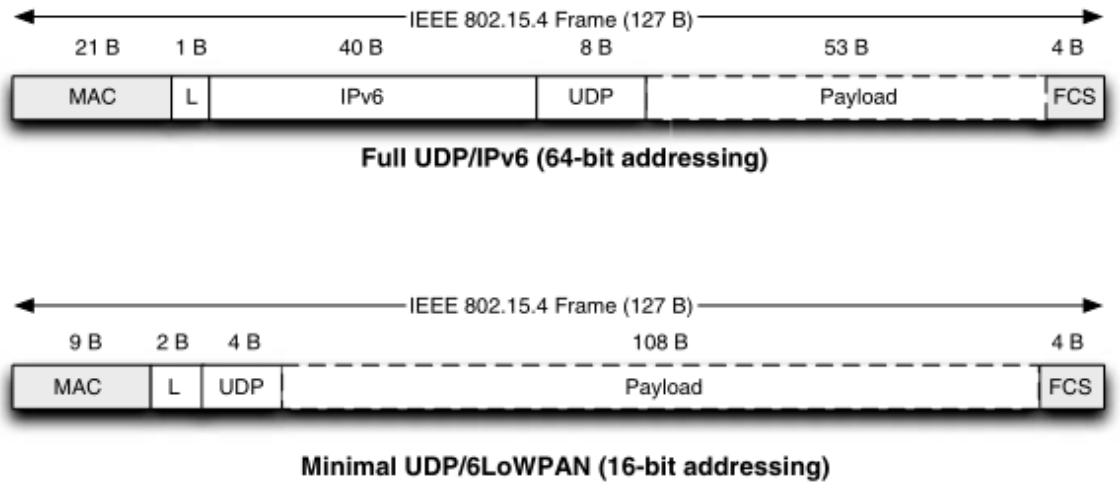


KUVA 1. OSI-malli, Wi-Fi-protokolla sekä 6LoWPAN-sovituskerros (3, s. 4.)

2.1 6LoWPAN-sovituskerroksen hyödyt

6LoWPAN-työryhmän päätavoite oli optimoida lähetettävien IPv6-pakettien kokoa, niin että ne olisivat sekä IPv6- että IEEE 802.15.4 -standardin mukaisia. IEEE 802.15.4 -standardin mukaan lähetettävän paketin maksimikoko on 127 tavua, kun taas IPv6-standardissa se on 1280 tavua. Ratkaisuna oli paketin jakaminen lähetysvaiheessa pienempiin osiin (fragmentointi) ja uudelleen kokoaminen joko jokaisen hypyn välein tai vasta määränpäässä. (3, s. 6–8.)

Toinen optimoinnin kohde oli lähetettävän paketin IPv6- ja UDP-otsikkotietojen pakkaus pienempään tilaan, jotta lähetettävän paketin hyötykuorma saataisiin kasvatettua mahdollisimman suureksi. Otsikkotietojen pakkaus perustuu laitteiden yhteisesti jaettuun tietoon verkosta sekä IPv6-osoitteen hierarkkiseen rakenteeseen, jonka takia IPv6-osoite voidaan yleensä jättää pois kokonaan paketin otsikkotiedoista. (1, s. 20–21.) Kuvassa 2 on esitetty esimerkki otsikkotietojen pakkauksella saavutettavaan hyötyyn.



KUVA 2. Otsikkotietojen pakkaus 6LoWPANin avulla (1, s. 21.)

6LoWPAN-verkon laitteet pystyvät myös luomaan automaattisesti oman verkko-osoitteensa 6LoWPAN ND -protokollan avulla (6LoWPAN Neighbor Discovery Protocol). Verkon laitteet kommunikoivat aluksi siirtokerroksella niiden laitteiden kanssa, jotka ne kuulevat omalla radioyhteydellään. Tässä vaiheessa asetetaan kanava-asetukset, mahdolliset salausavaimet sekä siirtokerroksen osoite. Kun yhteys on luotu yksittäisten laitteiden välillä, voidaan saman protokollan avulla muodostaa laitteille yksilöllinen IPv6-osoite. (1, s. 20–21.)

Pakettien reititys 6LoWPAN-verkossa tapahtuu RPL-protokollan avulla (Routing Protocol for Low-Power and Lossy Networks). RPL-protokolla perustuu matemaattiseen suuntavektorikuvaajamalliin (Directed Acyclic Graph), jonka tehtävänä on estää reitityssilmukat verkossa, joissa laitteen lähettämä viesti päätyisi takaisin itselleen. Kaikki laitteiden viestit ohjataan reunareitittimelle, joka toimii RPL-protokollan juurena (DODAG root). (4; 5, s. 10–11.)

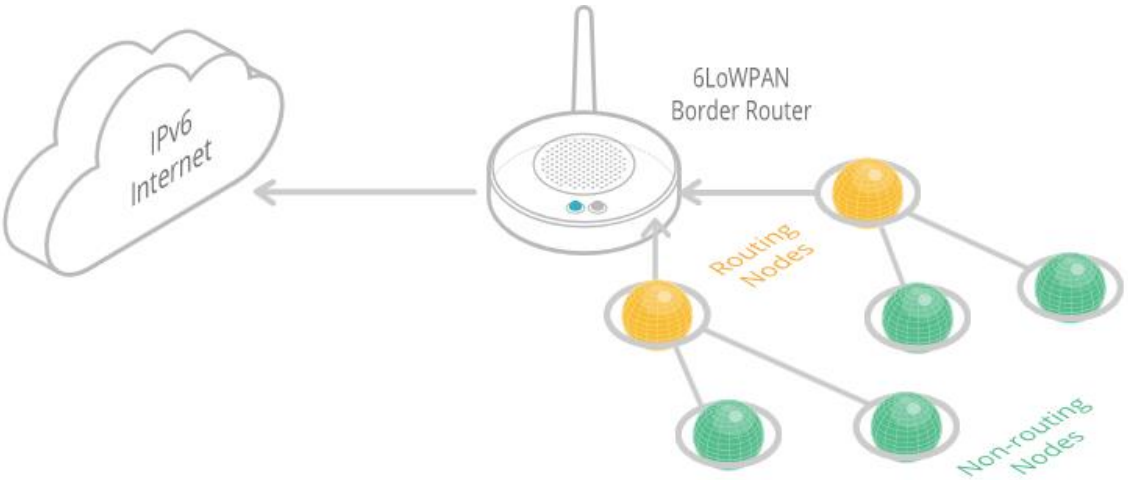
2.2 6LoWPAN-verkon laitteet

6LoWPAN-verkko koostuu tyypillisesti yhdestä reunareitittimestä sekä lukuisista noodeista, jotka voivat toimia reitittiminä, isäntinä tai nukkuvina isäntinä (kuva 3). Reunareititin (border router) toimii nimensä mukaisesti 6LoWPAN-verkon ja IPv6-verkon välissä. Sen tehtäviin kuuluu liikenteen välittäminen 6LoWPAN-verkon laitteiden ja internetin välillä, pakettien välittäminen 6LoWPAN-verkon sisällä sekä verkon muodostaminen ja ylläpito. (3, s. 2–3.)

Reitittävä noodi pystyy välittämään paketteja eteenpäin 6LoWPAN-verkossa muille laitteille ja sen radiolähetin on aina päällä eikä se nuku. Reitittäviä noodeja tarvitaan aina, jos halutaan muodostaan verkkomainen mesh-topologia, jossa verkon jokaisessa solmukohdassa on reitittävä noodi. (6.)

Isäntänä toimiva noodi ei pysty välittämään paketteja eteenpäin verkossa ja sen tulee olla aina yhteydessä reitittävään laitteeseen verkossa. Isäntä-noodi on yleensä myös aina päällä, koska reitittävät noodit eivät säilytä sille kuuluvia paketteja. (6.)

Nukkuva isäntä ei välitä paketteja verkossa ja tyypillisesti se nukkuu suurimman osan ajasta. Noodi herää tietyin väliajoin ajastimen tai ulkoisen keskeytyksen sattuessa, jolloin se suorittaa sille määrättyjä toimintoja, kuuntelee, onko sille tulossa paketteja reitittäviltä laitteilta, ja menee lopuksi takaisin nukkumaan säästääkseen virtaa. (6.)



KUVA 3. 6LoWPAN-sensoriverkon rakenne (7.)

3 ARM 6LOWPAN -OHJELMISTOPINO

ARM 6LoWPAN -ohjelmistopino on rakenteeltaan hyvin kevyt tapahtumapohjainen ohjelmistoympäristö, jota on mahdollista suorittaa ilman kolmannen osapuolen käyttöjärjestelmää. Muita 6LoWPAN-ohjelmistopinon suurimpia etuja ovat energiatehokkuus sekä alhaiset muistivaatimukset. 6LoWPAN-ohjelmistopino voidaan jakaa seuraaviin osiin:

- ohjelmistopinon ydin eli tapahtumapohjainen ympäristö
- protokollien moduulit
- valinnaiset turvallisuuskomponentit
- sovellusmoduulit.

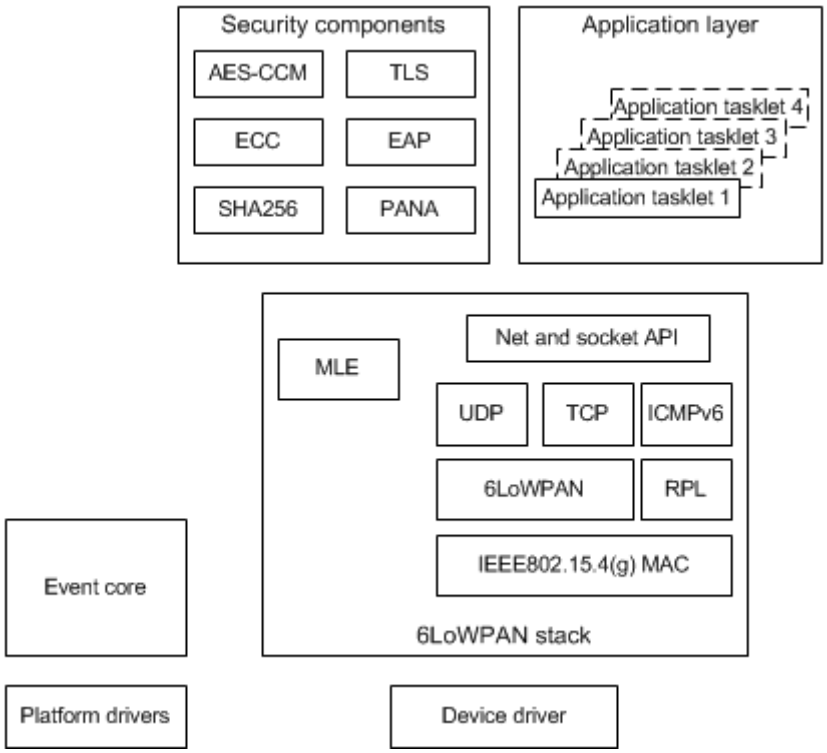
6LoWPAN-ohjelmistopinon ydin huolehtii alhaisen tason tapahtumista, aikataulutuksista sekä järjestelmän ajoitusfunktioista. Tämä ydinmoduuli huolehtii kaikesta perustoiminnallisuudesta, jota muut moduulit tarvitsevat. Ydinmoduuli tarvitsee toimiakseen jokaiselle alustalle erikseen käännetyt laiteajurit (6.)

6LoWPAN-ohjelmistopinosta löytyy laaja valikoima eri protokollia yksittäisinä moduuleina, jotka käyttävät sisäistä tietorakennetta datapakettien välittämiseen eteenpäin. Ohjelmistopinon modulaarisen rakenteen vuoksi tarpeettomia moduuleja voidaan jättää pois käännösvaiheessa ja näin saadaan laitteiden muistinkäyttöä pienemmäksi. Ylimpänä toimivan sokettimoduulin avulla sovellus voi lähettää ja vastaanottaa paketteja käyttäen IPv6-protokollan mukaista osoitetta ja porttinumeroa. (6.)

Ohjelmistopinosta löytyy myös tuki usealle erilaiselle salausmenetelmälle. Näiden avulla voidaan todentaa ja kryptata verkossa liikkuvaa data sekä hallinnoida laitteiden liittymistä verkkoon. Ohjelmistopino tukee seuraavia standardeja: PANA, EAP, TLS1.2, SHA-256, ECC sekä X509.3. (6.)

Ohjelmistopinon ylimmällä tasolla toimivan sovellusmoduulin toiminta perustuu tapahtumakäsittelijään, jota kutsutaan taskletiksi. Sovelluskehittäjä voi kehittää useita eri taskletteja, joilla kaikilla on täysi pääsy ohjelmistopinon ominaisuuksiin. Sovelluksen toiminnassa taskletti saa vastaan tapahtuman, jonka se prosessoi ja suorittaa vaaditun toimenpiteen, kuten esimerkiksi lukee sensorin arvon, rakentaa paketin ja lähettää sen, asettaa uuden ajastetun tapahtuman ja palauttaa

hallinnan ohjelmistoytimelle, joka huolehtii varsinaisesta paketin lähetyksestä. (6.) Kuvassa 4 näkyy 6LoWPAN-ohjelmistopinon modulaarinen rakenne.



KUVA 4. 6LoWPAN-ohjelmistoytimen rakenne (6.)

4 SIMULAATTORI

Simulaattori on yrityksen sisäisesti kehittämä työkalu sensoriverkon mallintamiseen ohjelmallisesti käyttäen 6LoWPAN-ohjelmistopinoa. Simulaattori on kirjoitettu C++:lla ja kaikki simulaattorin luokat on saatu Python-ympäristön käytettäväksi SWIG-työkalun avulla. Simulaattoria on näin helppo konfiguroida ja suorittaa Python-skriptien avulla. Simulaattori toimii Linux-ympäristössä Python 2.7 -version kanssa. (8.)

4.1 Simulaation ajaminen

Simulaation ajaminen Python-skriptin avulla vaatii kuvassa 5 näkyvät toimenpiteet (8):

1. ajastin-luokan alustaminen
2. käytettävän kanavamallin alustaminen
3. simuloitujen noodien lisäys verkkoon
4. noodien alustaminen
5. simulaation käynnistäminen.

```
1  from NanoSimulator import *
2
3  # Init scheduler
4  s = Scheduler()
5
6  # Create channel model
7  model = LogDistanceModel(-90.0, -100.0, 6.0, 10.0)
8  s.net.add(model)
9
10 # Initialize router
11 router = IPCmessageNode("applications/NanoRouter3")
12 s.net.add(router)
13 router.init()
14
15 # Initialize node
16 node = IPCmessageNode("applications/Node")
17 s.net.add(node)
18 node.init()
19
20 # simulate 20 seconds
21 second = 1000000
22 s.run(20*second)
```

KUVA 5. Esimerkkikoodi simulaation ajamisesta Pythonilla (8.)

4.2 Ajastin-luokka

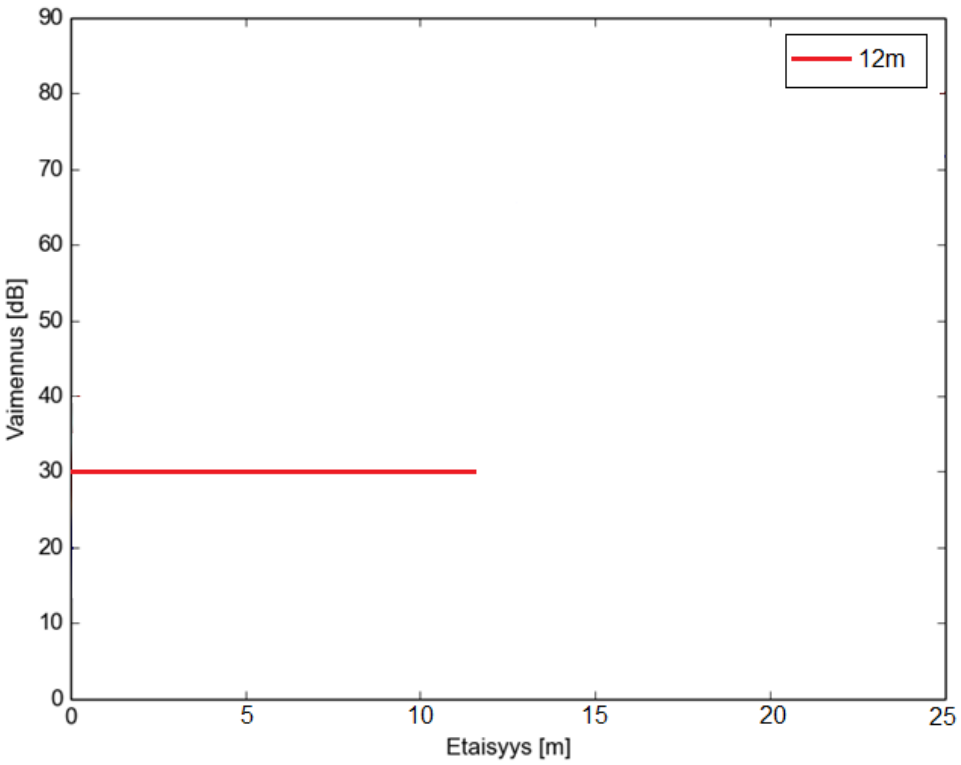
Ajastin-luokka on simulaattorin ydinkomponentti, joka huolehtii kaikkien simuloitujen komponenttien synkronoinnista keskenään. Simulaattorin aikayksikkönä käytetään yhtä mikrosekuntia. Ajastin-luokka kiihdyttää simulaation nopeutta hyppäämällä aina seuraavaan verkkotapahtumaan eikä toimi reaaliaikaisesti. (8.)

4.3 Kanavamallit

Simulaattorissa on käytettävissä kolme kanavamallia radiotien mallinnukseen: etäisyysmalli, Friisin kanavamalli sekä logaritminen vaimennusmalli (8). Opinnäytetyön aikana käytettiin pääsääntöisesti etäisyysmallia simulaatioissa, jotta kanavan vaimennuksen aiheuttamat mahdolliset ongelmat voitiin sulkea pois kehitystyön aikaisessa vaiheessa.

4.3.1 Etäisyysmalli

Etäisyysmalli on kaikista yksinkertaisin ja nopein käyttää simulaatiossa. Etäisyysmalli ottaa huomioon vain verkossa olevien noodien etäisyyden toisiinsa päätellessään lähetyksen onnistumista eikä se ota huomioon radiotiellä tapahtuvaa vaimennusta. Kaikki liikenne määriteltyä etäisyyttä kauempana menetetään automaattisesti kuvan 6 mukaisesti. (8.)



KUVA 6. Kanavan vaimennus etäisyysmallilla (8.)

4.3.2 Friisin kanavamalli

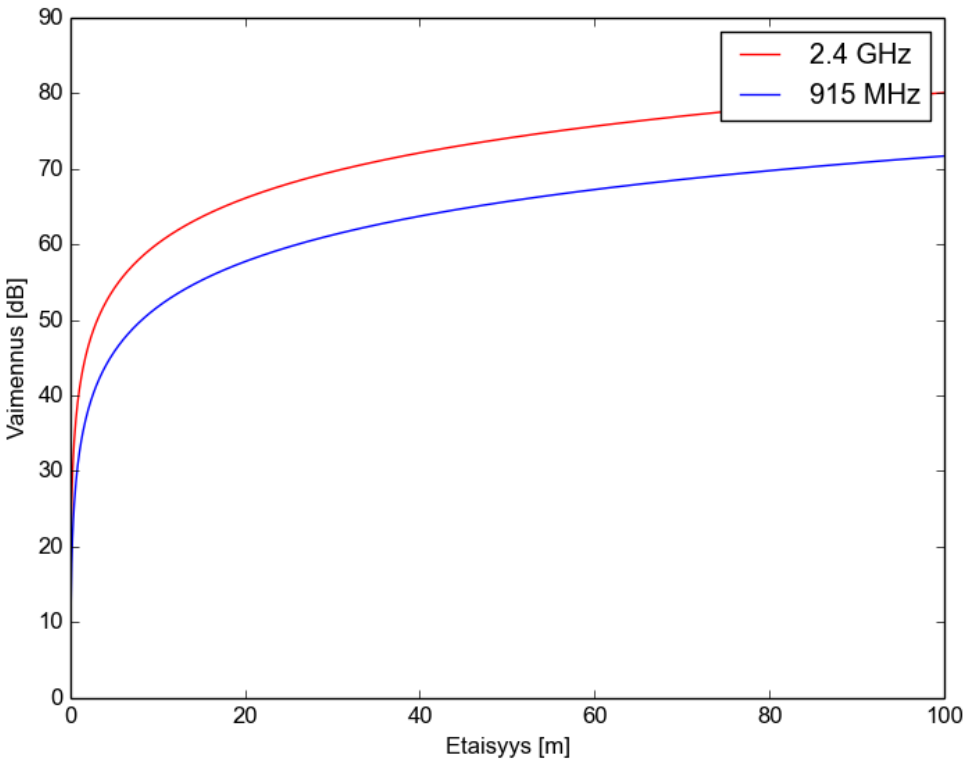
Friisin kanavamalli kuvaa vapaassa tilassa tapahtuvaa kanavan vaimennusta. Tyypillinen kanavan vaimennus ulkotilassa eri taajuuksilla näkyy kuvassa 7. Friisin kanavamalli lasketaan kaavan 1 mukaisesti (8):

$$L = 20 * log_{10}\left(\frac{4\pi d}{\lambda}\right) \qquad \textit{KAAVA 1}$$

L = kanavan vaimennus

d = etäisyys metreissä

λ = aallonpituus



KUVA 7. Kanavan vaimennus Friisin mallilla (8.)

4.3.3 Logaritminen vaimennusmalli

Logaritminen vaimennusmalli ennustaa radiotien vaimennusta sisätiloissa tai tiheästi asutetuilla alueilla. Kuvassa 8 näkyy erilaisten rakenteiden aiheuttamia muutoksia kanavan vaimennuksessa. Logaritminen vaimennus lasketaan kaavan 2 avulla seuraavasti (8):

$$L = L_0 + 10 * \gamma * log_{10}\left(\frac{d}{d_0}\right) + X_g \qquad \text{KAAVA 2}$$

L = kanavan vaimennus

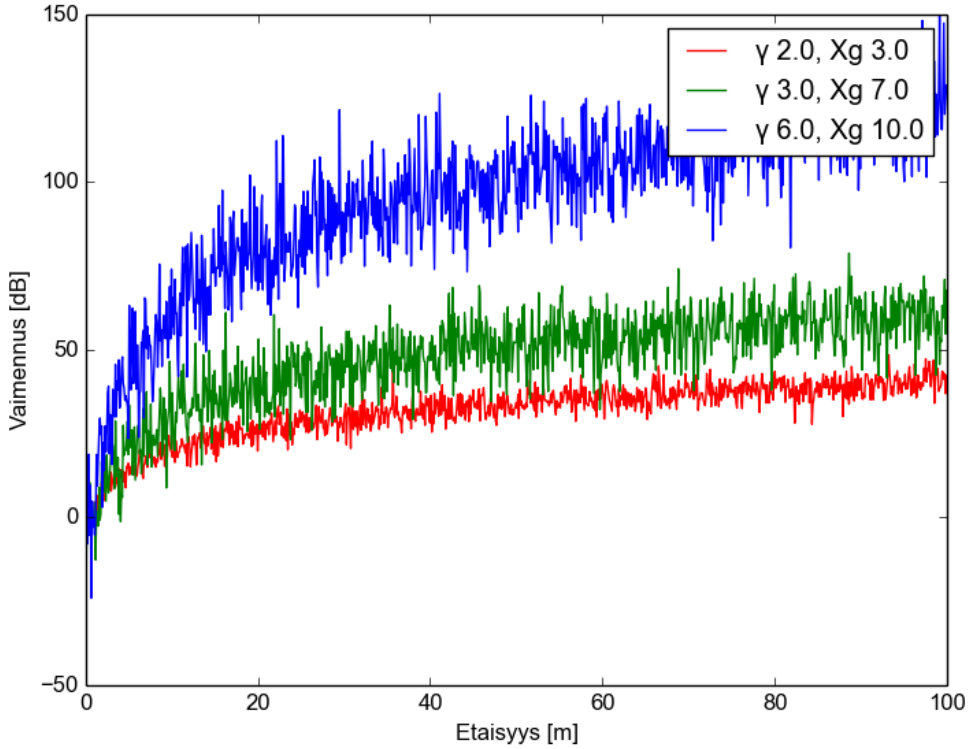
L_0 = vaimennus vertailuetäisyydellä

d = etäisyys metreissä

d_0 = vertailuetäisyys

γ = kanavan vaimennuseksponentti, empiirisesti mitattu

X_g = taustakohina



KUVA 8. Kanavan vaimennus logaritmisella mallilla (8.)

4.4 Laitteiden lisäys simulaatioon

Simulaattorissa on mahdollista lisätä kolme erityyppistä laitetta eli noodia verkkoon:

1. IPC-noodi
2. Wireshark-noodi
3. PCAP-noodi.

IPC-noodille annetaan käynnistyksen yhteydessä parametreina konfiguraatiotiedosto, joka sisältää tarvittavat verkkoparametrit. IPC-noodille voi myös määrittää lokitiedoston nimen, jonne tulevat kaikki ohjelmistopinon viestit simulaation ajalta. (8.)

Wireshark-noodi avaa nimensä mukaisesti taustalle Wireshark-ohjelman ja lähettää kaikki kuulemansa viestit radiotieltä suoraan ohjelmaan. Näin verkkoliikennettä voidaan seurata suoraan simulaation edetessä. PCAP-noodi toimii vastaavasti mutta tallentaa kaikki kuulemansa viestit suoraan pcap-tiedostoon, jonka nimi tulee antaa parametrina. (8.)

Simulaatioon lisättäville laitteille tulee myös antaa sijainti xy-koordinaatistossa ennen alustamista, jotta simulaattori voi laskea kanavan vaimennuksen jokaiselle laitteelle. Simulaattorissa etäisyy-

den yksikkönä käytetään yhtä metriä. (8.) Yleensä reunareititin asetetaan origoon (0, 0) ja seuraavat noodit tietyn välimatkan päähän reunareitittimestä, esimerkiksi pisteisiin (10, 0), (20, 0) ja niin edelleen. Näin saadaan myös luotua helposti hyppyjä simulaatioon käyttämällä esimerkiksi etäisyysmallia 12 metriä, koska kaikki noodit eivät kuule toisiaan.

4.5 Simulaattorin hyödyt

Simulaattorin suurimpia hyötyjä testauksessa verrattuna fyysisiin laitteisiin ovat sen nopeus, edullisuus, monipuolisuus sekä tulosten tarkastelu. Simulaattorin avulla vältetään fyysisten laitteiden uudelleenohjelmointi aina tehtyjen muutosten jälkeen ja sensoriverkkoon voidaan lisätä huomattavasti enemmän laitteita kuin aiemmin. Satojen noodien keskinäistä toimintaa voidaan mallintaa yhdellä tietokoneella, eikä yrityksen tarvitse ostaa testausta varten erikseen lukuisia fyysisiä laitteita.

Simuloidut laitteet kirjoittavat lokitiedostoon kaikki tapahtumat sekä simulaattori tallentaa kaikki verkossa lähetetyt paketit yhteen Wireshark-lokitiedostoon. Laitteiden mahdollisia kaatumisia ja muistivuotoja voidaan tarkastella tarkemmin GDB:n ja Valgrindin kaltaisten virheidenetsintätyökalujen avulla.

5 CLITEST-TESTAUSKEHYS

Clitest-testauskehys on ARMin sisäisesti kehittämä Python-pohjainen testaustyökalu, jonka avulla voidaan välittää komentoja testattavalle laitteelle (DUT) ja suorittaa helposti testiskriptejä. Testattava laite voi olla fyysinen tai simuloitu noodi tai muu ohjelma, kunhan se on toteutettu komentorivipohjaisesti. (9.)

5.1 Testauskehyksen toiminta

Testauskehys käynnistetään komentoriviltä, missä sille annetaan argumentteina vähintään suoritettava yksittäinen testiskripti tai laajemman testattavan kokoelman nimi. Yleensä komentoriviltä kerrotaan myös, onko testattava laitetyyppi fyysinen vai simuloitu noodi tai jokin muu ohjelmaprosessi. Wireshark-prosessi pitää käynnistää komentoriviltä niissä testeissä, joissa verifioidaan sensoriverkossa liikkuvien pakettien sisältöä. Laajemmassa automaatiotestauksessa käytetään myös yleisesti Valgrind-työkalua, joka etsii testattavasta sovelluksesta muistivuotoja ja muita virheitä.

Testauskehys lukee jokaisen testiskriptin metatiedoista testin nimen, mahdollisen testikokoelman, testattavien laitteiden määrän, tyypin sekä simulaatiotesteissä laitteiden sijainnin xy-koordinaatistossa. Tämän jälkeen testauskehys suorittaa varsinaisessa testiskriptissä laitteiden ylösajon, varsinaisen testin sekä lopuksi laitteiden alasajon ennen siirtymistä seuraavaan testiin. Jokaisesta testistä tallennetaan omaan kansioon lokitiedostot laitteiden toiminnasta testin aikana sekä mahdollinen Wireshark-loki, josta löytyy testin aikana kaapattu sensoriverkon verkkoliikenne.

5.2 Testiskriptit

Testiskriptit on myös kirjoitettu Pythonilla ja niillä pyritään testaamaan täsmällisesti yhtä ohjelmistopinon ominaisuutta kerrallaan. Testiskriptiin kuuluu aina yksi Python-luokka, joka jaetaan neljään funktioon: testin alustus, ylösajo, itse testin suoritus sekä testin alasajo.

Testin alustusvaiheessa annetaan kaikki metatiedot, jotta testauskehys osaa luokitella ja suorittaa testin oikealla tavalla. Ylösajovaiheessa testattavat laitteet konfiguroidaan halutulla tavalla ja käynnistetään verkkoon liittyminen. Suoritusvaiheessa tehdään itse testi, jolla yleensä tarkistetaan sensoriverkossa liikkuvien pakettien oikeaa reititystä laitteelta toiselle tai sitten varmistetaan yksittäisten pakettien sisältöä. Jos testin aikana noodi poistuu verkosta tai ei vastaa halutulla tavalla, testin suoritus lopetetaan ja testauskehys palauttaa virheen. Testin viimeisessä alasajovaiheessa laitteet sammutetaan hallitusti ja palautetaan lopputulos käyttäjälle.

6 CLIAPP-TESTAUSSOVELLUS

Opinnäytetyössä käytettiin hyväksi C:llä kirjoitettua Cliapp-testaussovellusta, joka alun perin kehitettiin Thread-protokollan testaukseen. Thread on 6LoWPANia käyttävä verkkoprotokolla, jolla pyritään yhdistämään kodin sähkölaitteet internetiin (10). Vaikka Thread on osa ARM 6LoWPAN -ohjelmistopinoa, ei se ole suoraan yhteensopiva normaalin 6LoWPAN-sensoriverkon kanssa.

Testaussovellus on komentorivipohjainen ohjelma, jonka avulla ohjelmistopino voidaan konfiguroida halutulla tavalla ennen käynnistystä, esimerkiksi asettaa se käyttämään Thread- tai 6LoWPAN-protokollaa. Muita tyypillisiä komentoja ennen laitteen käynnistämistä ovat MAC-osoitteen asettaminen sekä mahdollisen verkkoliikenteen salauksen käyttöönotto. Sovelluksen suorituksen aikana voidaan myös muuttaa useita eri parametreja ja testata niiden vaikutuksia sensoriverkon toimintaan. Sovelluksen avulla voidaan myös lähettää ping-komentoja ja UDP-dataa suoraan laitteesta toiselle ja näin testata eri viestien kulkemista laitteelta toiselle.

Opinnäytetyön alussa testaussovellusta käytettiin lähinnä fyysisten laitteiden testaukseen testauskehyksen avulla, joka lähettää komennot testiskriptien avulla, tai sitten testaaja kirjoitti manuaalisesti komentoja laitteelle komentoriviltä USB-sarjaportin välityksellä. Opinnäytetyön aikana simulaattoriin lisättiin tuki tämän sovelluksen suorittamiselle ja minun tehtäväkseni jäi lisätä 6LoWPAN-tuki siihen.

7 OPINNÄYTETYÖN TOTEUTUS

Opinnäytetyön suoritus jakaantui karkeasti seuraavaan neljään osaan: vaatimusmäärittelyn luominen, simulaattorin yhdistäminen testauskehyksen kanssa, testaussovelluksen laajentaminen sekä ensimmäisten automaattisten testiskriptien luominen 6LoWPAN-ohjelmistopinolle.

7.1 Vaatimusmäärittely

Opinnäytetyön ensimmäisessä vaiheessa kerättiin yrityksessä työskenteleviltä insinööreiltä ideoita simulaattorin jatkokehitykseen testauksen näkökannalta. Kaikki saadut ideat kerättiin yhteen ja jaettiin tärkeysjärjestykseen simulaattorin kehittäjän kanssa asteikolla korkea, keskisuuri tai matala prioriteetti. Nämä tehtävät jaettiin sitten tasaisesti useammalle saman ryhmän työntekijälle.

7.1.1 Korkean prioriteetin tehtävät

Ensimmäiseksi simulaattori piti saada yhdistettyä testauskehyksen kanssa siten, että simulaattorille voitaisiin lähettää ja vastaanottaa komentoja standardin syöttö- ja tulostusvirran kautta. Simulaattori käynnistää myös vakiona jokaisen simuloidun laitteen erilliseen komentoikkunaan, jonka toiminta piti muuttaa vaihtoehtoiseksi. Jotta ohjelmistopinoa voisi testata tehokkaasti, pitää simulaattorin osata käyttää valmiiksi käännettyjä kirjastokomponentteja sen sijaan, että simulaattori kääntää aina kaikki kirjastot itse.

Muita tärkeitä kehityskohteita olivat tuki usealle reunareitittimelle samassa simuloidussa verkossa sekä radiotaajuuskanavien käyttöönotto, koska nykytilassa simuloidut noodit kuulevat kaiken liikenteen omalla kuuluvuusalueellaan eivätkä välitä käytettävästä kanavasta. Tämän jälkeen voitaisiin kehittää myös simulaatiotuki taajuushyppelylle sensoriverkossa.

7.1.2 Keskisuuren prioriteetin tehtävät

Simulaattorin ajastin-luokkaa tulisi kehittää siten, että se voisi toimia myös reaaliajassa eikä kiihdytetysti hyppien aina tapahtumasta seuraavaan. Tähän tarvittaisiin vastaava ajastin, joka löytyy fyysisistä laitteista, jonka tarkkuuden tulisi olla noin yksi millisekunti. Näin simuloitu noodi voisi

keskustella simulaattorin ulkoisen verkon kanssa ja esimerkiksi rekisteröityä verkkopalvelimelle asti, jonka kautta sille voitaisiin välittää yksinkertaisia komentoja.

Muita tämän tason vaatimuksia olivat aikaleimojen lisääminen ohjelmistopinon tulostuksille virheidenetsintää helpottamaan sekä dynaamisen muistin määrän rajoittaminen sovellukselle ja 6LoWPAN-ohjelmistopinolle, jotta voitaisiin tutkia paremmin laitteiden muistivaatimuksia eri konfiguraatioissa.

7.1.3 Matalan prioriteetin tehtävät

Tämän tason tehtävät olivat sellaisia ominaisuuksia, jotka eivät olleet kriittisiä opinnäytetyön aikana mutta olisivat muuten hyviä kehityskohteita simulaattoriin tulevaisuudessa.

Langattoman ohjelmistopäivityksen siirtämistä kaikille verkon laitteille pitäisi pystyä simuloimaan. Verkkoliikenteen lokitiedostoista tulisi piirtää Wiresharkin avulla automaattisesti kuvaajia eri protokollien liikennemääristä. Lisäksi useampaa simulaattori-instanssia pitäisi pystyä suorittamaan rinnakkain, jotta simuloidun verkon kokoa saataisiin skaalattua suuremmaksi.

7.2 Simulaattorin yhdistäminen testauskehykseen

Ensimmäisessä toteutusvaiheessa simulaattorin IPC-noodi-luokkaan tehtiin uusi muodostin, jossa yhtenä parametrina annetaan boolean-tyyppinen muuttuja, jonka perusteella C++-luokalle kerrotaan, käynnistääkö simulaattori lapsiprosessin itse. Muut muodostimelle välitettävät parametrit ovat suoritettavan tiedoston nimi sekä käytettävä konfiguraatiotiedosto. Lisäksi muodostimessa suoritettavan prosessin nimi tallennetaan string-muodossa muistiin, jotta Pythonin automaattinen roskienkerääjä ei hävitä niitä vahingossa.

Seuraavana työvaiheena oli laajentaa SWIG-työkalun rajapintaa lisäämällä sinne yksi Python-funktio, jonka avulla voidaan välittää testauskehykselle simuloidun laitteen parametrit: prosessin nimi, ID-numero, konfiguraatiotiedoston nimi sekä viestijonojen nimet simuloidun laitteen sekä simulaattorin ytimen välillä. Tämän jälkeen testauskehyksen kautta voitiin käynnistää simuloitu noodi aliprosessina ja välittää komentoja syöttö- ja tulostusvirran kautta. Näissä kahdessa edelli-

sessä työvaiheessa suurin vastuu oli simulaattorin kehittäjällä mutta osallistuin aktiivisesti mukaan kehitys- ja testaustyöhön.

Kun simulaattoriin tehdyt muutokset olivat valmiit, alkoi testauskehyksen ylläpitäjä lisätä tukea simuloidun Thread-protokollaa käyttävän noodin suorittamiseksi Python-testiskripteillä. Minun tehtävänäni oli testata tehtyjä muutoksia, ja saimmekin melko nopeasti simulaattorin keskustelemaan yhdessä testauskehyksen kanssa.

7.3 Testaussovelluksen laajentaminen

Testaussovelluksen laajentaminen jaettiin kolmeen erilliseen vaiheeseen: reunareitittimen lisäys, reitittävän noodin lisäys sekä tuki yleisimmille salausmenetelmille. Kehitystyön pohjana toimi simulaattorin aiemmin käyttämä sovellus, jonka vaatimat asetukset luettiin erillisestä konfiguraatiotiedostosta. Kaikki nämä asetukset tuli siirtää testaussovelluksen sisälle ja tärkeimpiä asetuksia piti voida muuttaa komentorivin kautta.

7.3.1 Reunareititin

6LoWPAN-reunareititin sisältää eniten toiminnallisuutta 6LoWPAN-verkon laitteista. Muun muassa seuraavat asiat piti alustaa sovelluksessa ennen reunareitittimen käynnistyskomentojen suorittamista:

- ND-protokollan asetukset
- RPL-protokollan asetukset
- RPL DODAG -asetukset
- kuljetuskerroksen salausasetukset
- Globaalin IPv6-verkon prefiksi
- sensoriverkon nimi.

6LoWPAN-asetusten alustuksen jälkeen luotiin alias ”6dut1”, jonka avulla laite voidaan yhden komennon avulla konfiguroida toimimaan reunareitittimenä. Seuraavaksi verkon käynnistyskomentoon lisättiin seuraavat funktiot reunareitittimelle:

- salausmenetelmän valitseminen (pois päältä tässä vaiheessa)
- reunareitittimen alustaminen

- RPL DODAG -instanssin alustaminen
- RPL DODAG -prefiksin päivittäminen
- reunareitittimen kontekstin päivittäminen
- RPL DODAG -reittien päivittäminen
- sensoriverkon rajapinnan käynnistyskomento
- RPL DODAG -instanssin käynnistäminen.

Jokaisesta funktiosta tarkistettiin, että ohjelmistoydin palauttaa kutsun jälkeen paluuarvon nolla, joka tarkoittaa virheetöntä suoritusta, ennen kuin siirryttiin käynnistyksessä eteenpäin. Tässä vaiheessa löytyi ensimmäinen isompi ohjelmointivirhe ohjelmistoytimestä, koska tapahtumakäsittelijä ei koskaan palauttanut sovellukselle palautuskoodia onnistuneesta käynnistyksestä ja sovellus jäi jumiin odottamaan vastausta. Virhe raportoitiin eteenpäin ja sovellusta muokattiin väliaikaisesti siten, ettei palautuskoodia jääty odottamaan ja sovelluksen suoritusta jatkettiin normaalisti eteenpäin.

Muutoksia testattiin käynnistämällä testaussovelluksen reunareititin sekä vanhempi staattisesti konfiguroitu reitittävä noodi samaan instanssiin Python-skriptillä ja varmistettiin, että noodi liittyy reunareitittimeen ongelmitta. Lisäksi testattiin, että aikaisemmin lisätty tuki TUN-rajapinnalle toimii myös 6LoWPAN-reunareitittimen kanssa. Tämä tehtiin käynnistämällä simulaatio TUN-rajapinnan kanssa, jonka jälkeen Linuxin komentoriviltä lähetettiin ping-komentoja sensoriverkon laitteille. Testauksen päätyttyä muutokset lähetettiin versiohallintaan muiden insinöörien arvioitavaksi ja lopulta koodi sulautettiin osaksi testaussovelluksen päähaaraan.

7.3.2 Reitittävä noodi

Reitittävän noodin lisääminen testaussovellukseen oli huomattavasti yksinkertaisempi prosessi, koska suurin osa työstä tehtiin jo reunareitittimen lisäyksen aikana. Sovellukseen lisättiin kaksi aliasta ”6dut2” ja ”6dut3”, jotka asettavat laitteet toimimaan reitittävinä noodeina. Verkon käynnistämiskomentoon piti lisätä noodille vain kaksi funktiokutsua, joilla määriteltiin kanava-asetukset, yhteyden skannausaika sekä verkon nimi, johon halutaan liittyä. Muita pienempiä muutoksia olivat multicast-asetusten siirto omaan tietueeseen sekä uusi komentorivikutsu verkon skannausajan säätämistä varten.

Testaus suoritettiin samalla tavalla kuin reunareitittimen kanssa: vanhempi staattisesti konfiguroitu reunareititin käynnistettiin yhdessä uudemman testaussovelluksen noodin kanssa ja varmistettiin, että noodi liittyy verkkoon ja laitteet vastaavat toistensa lähettämiin ping-komentoihin. Testauksen aikana löytyi uusi ohjelmointivirhe ohjelmistoytimestä, missä noodit pysyivät pitkään verkossa vain ollessaan suorassa yhteydessä reunareitittimeen mutta menettivät yhteyden, jos ne olivat liittyneet verkkoon toistensa välityksellä useamman hypyn päästä reunareitittimestä.

Kun muutokset oli saatu sulautettua sisään versiohallintaan, järjesti järjestelmätestauksesta vastaava insinööri meille testaajille kaksi koulutussessiota, joista ensimmäisessä harjoiteltiin testauskehyksen ja simulaattorin käyttämistä 6LoWPAN-reunareitittimen ja -noodin kanssa ja toisessa sessiossa sama 6LoWPAN-harjoitus toistettiin sulautettujen ARM-pohjaisten kehitysalustojen kanssa. Näissä sessioissa löydettiin sekä testauskehyksestä että itse testaussovelluksesta muutamia pienempiä ohjelmointivirheitä, jotka raportoitiin korjausta varten.

7.3.3 Salausmenetelmien lisäys

Testaussovellukseen lisättiin tuki kuljetuskerroksen PSK-salaukselle (Pre-shared Key) sekä PANA-PSK-todennukselle (Protocol for Carrying Authentication for Network Access). PANA-todennuksesta tuetaan myös ECC- ja ECC+PSK-muotoja (Elliptic Curve Cryptography), mutta niitä ei sisällytetty tähän vaiheeseen vielä hiukan työläämmän toteutuksen vuoksi. Ensimmäiseksi sovellukseen lisättiin uusi tietue 16 merkkiä pitkille salausavaimille sekä lyhyemmille salausavaimien tunnuksille. Seuraavaksi luotiin komentorivikutsut, joilla voitiin ottaa haluttu salausmenetelmä käyttöön ja asettaa salausavaimet ja niitä vastaavat tunnukset. Sovelluksen alustusvaiheessa avaimille ja tunnuksille asetettiin myös oletusarvot.

Itse laitteen käynnistyessä tarkistettiin ensin, onko PSK- tai PANA-PSK-tila otettu käyttöön, ja sen mukaan asetettiin omilla funktioilla avaimet ja tunnukset käyttöön. PANA-PSK-muodossa reunareititin käynnistää myös PANA-palvelimen ja noodi vastaavasti PANA-asiakasohjelman. Testaus tehtiin vastaavalla tavalla kuin aikaisemmissa vaiheissakin eli kokeiltiin, että salaukset toimivat ristiin samoilla salausavaimilla vanhemman staattisen simulaattorisovelluksen ja uudemman testaussovelluksen välillä.

7.4 Automaattisten testien luominen

Viimeisessä vaiheessa kirjoitettiin ensimmäiset simuloidut 6LoWPAN-testiskriptit, joita voitiin suorittaa automaattisesti jatkuvassa integrointiympäristössä. Ehdin kirjoittaa valmiiksi kolme testiä, joissa verkko rakennetaan kahdella tai kolmella laitteella hiukan erilaista topologiaa käyttäen ja laitteiden välisen yhteyden toimivuus varmistettiin ping-komentojen avulla, jonka jälkeen laitteet sammutetaan ja testin suoritus päättyy. Monimutkaisempia testejä ei ollut mahdollista kirjoittaa vielä tässä vaiheessa, koska laitteiden muodostamat hypyt verkossa toimivat vielä epäluotettavasti. Testien suoritus tapahtui komentoriviltä kutsumalla testauskehystä sovelluksen päähakemistossa ja antamalla sille argumentteina vähintään testiskriptin nimi sekä testityyppinä simulaatio, esimerkiksi:

```
python clitest.py --tc 6lowpan_test_ping_2_nodes --type simulate
```

Skriptin alustusvaiheessa annettiin metatiedoissa testin nimi, joka on sama kuin itse Python-tiedoston nimi. Muita vaadittuja metatietoja olivat otsikkotiedot, joissa testiä ja sen tarkoitusta kuvataan lyhyesti. Vaatimuksiin piti lisätä testattavien laitteiden määrä, niiden nimet sekä oletusarvona käytettävä laitetyyppi: fyysinen, simuloitu tai prosessi. Kun testattavat laitteet olivat simuloituja, niiden sijainnin pystyi myös määrittelemään erikseen xy-koordinaatistossa.

Testin ylösajovaiheessa laitteet konfiguroitiin käyttäen aikaisemmin luotuja aliaksia: ”6dut1” reunareitittimelle sekä ”6dut2” ja ”6dut3” reitittäville noodeille. Tämän jälkeen laitteet käynnistettiin ja odotettiin, että laitteet rekisteröityvät verkkoon onnistuneesti, jonka jälkeen siirryttiin testin suoritusvaiheeseen. Tässä vaiheessa laitteilta kysyttiin ensin niiden paikalliset ja globaalit osoitteet läpi, joihin sitten lähetettiin ping-komentoja ristiin kaikilta laitteilta. Jos yksikään ping-komento ei saanut vastausta, lopetettiin testin suoritus ja palautettiin virhe komentoriville mutta yleensä testin suoritusvaiheen mentyä läpi normaalisti siirryttiin alasajovaiheeseen, jossa laitteet sammutettiin ja testin suoritus päättyi.

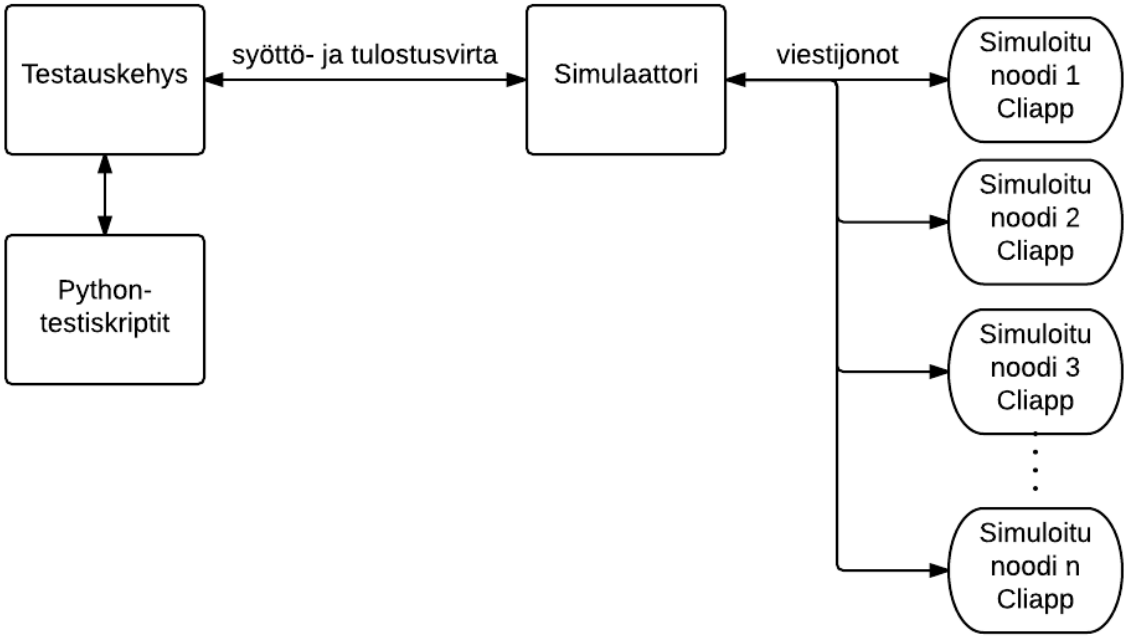
7.5 Tulosten tarkastelu

Simulaattori saatiin yhdistettyä Python-pohjaiseen testauskehykseen ja sen kautta koko muuhun testausjärjestelmään ja lopputuloksena oli kuvan 9 mukainen testausjärjestelmä. Yhdistäminen oli

varsin yksinkertainen prosessi, koska simulaattorin C++-luokkia ja -funktioita voitiin jo kutsua Python-skriptien kautta. SWIG-työkalu loi tämän yhteyden ohjelmointikielien välille automaattisesti simulaattorin käännösvaiheessa. SWIG-työkalun rajapintaa piti vain hiukan laajentaa, jotta testauskehykselle voitiin välittää simuloidun noodin parametrit ja noodi voitiin näin käynnistää aliprosessina testauskehyksen kautta.

Simulaattorin alkuperäisesti käyttämä sovellus ei sopinut kovin hyvin testaukseen, koska sille ei voinut välittää komentoja suorituksen aikana ja kaikki asetukset luettiin erillisestä konfiguraatiotiedostosta. Tämän vuoksi päätettiin laajentaa komentorivipohjaista Cliapp-testaussovellusta lisäämällä siihen tuki 6LoWPAN-protokollalle. Sovellukseen lisättiin tuki reunareitittimelle ja reitittävälle noodille, yleisesti käytetyt salausmenetelmät sekä komentoriviparametrit asetusten muokkaamiseksi ohjelman suorituksen aikana.

Viimeisessä työvaiheessa luotiin ensimmäiset Python-testiskriptit 6LoWPAN-laitteiden simuloitua testausta varten. Testiskripti kutsuu testauskehystä kertoen, että kyseessä on simuloitu testi, jolloin testauskehys käynnistää aliprosessina simuloidun Cliapp-sovelluksen, jonka jälkeen simuloitu noodi konfiguroidaan testiskriptin komennoilla ja suoritetaan itse testi.



KUVA 9. Yleiskuva valmiista testausjärjestelmästä

Opinnäytetyössä saavutettiin pääsääntöisesti alussa asetetut tavoitteet. Projektin kehitystyöhön osallistui useampi insinööri ja minä tein omalta osaltani annetut tehtävät. Simulointiin perustuva

testausjärjestelmä otettiin valmistuttuaan nopeasti mukaan jatkuvaan käännös- ja integrointiympäristöön ja simuloituja testejä suoritettiin ennen ja jälkeen koodimuutosten sulauttamista versiohallintaan. Pääasiallinen testaaminen on saatu muutettua jatkuvan integrointiympäristön avulla täysin automaattiseksi. Tämä on ollut suuri kehitysaskel yrityksen testausjärjestelmässä.

Kaikkia vaatimusmäärittelyssä asetettuja tavoitteita ei ehditty saavuttaa opinnäytetyön aikana mutta näitä tavoitteita useampi insinööri on kehittänyt vähitellen opinnäytetyön jälkeen. Itse olen osallistunut tähän työhön kirjoittamalla suuren määrän testiskriptejä, joilla testataan laajemmin ohjelmistopinon ominaisuuksia. Olen myös kirjoittanut Cliapp-testaussovellukseen useita lisäominaisuuksia testausta varten.

8 YHTEENVETO

Opinnäytetyön tavoitteena oli kehittää ja automatisoida ARM 6LoWPAN -ohjelmistopinon testausta simuloimalla sensoriverkon toimintaa, rakennetta ja eri laitteita. Työssä hyödynnettiin erikseen kehitettyä simulaattoria, Clitest-testauskehystä sekä Cliapp-testaussovellusta, jotka piti saada toimimaan yhdessä, jotta automaattinen testaus olisi mahdollisimman helppoa tulevaisuudessa.

Vaatimusmäärittelyn perusteella ensimmäiseksi luotiin yhteys simulaattorin ja testauskehyksen välille, jotta simulaation voisi käynnistää testauskehyksen kautta. Tämän jälkeen testaussovellukseen luotiin tuki 6LoWPAN-protokollan laitteille, joiden toimintaa voitiin sitten testata automaattisesti testiskriptien avulla.

Työtä tehdessäni opin paljon ohjelmistokehityksestä ryhmässä, jossa jokainen jäsen osallistui projektiin kehittämällä tiettyä komponenttia, joiden summana saatiin lopputulokseksi automaattisesti toimiva simuloitu testausjärjestelmä. Versiohallinnan käyttäminen projektissa oli uutta minulle, ja useiden insinöörien tehdessä muutoksia samaan tiedostoon oli minulla vaikeuksia sulauttaa kaikki muutokset yhteen ilman konflikteja. Itse testaussovelluksen laajentaminen sujui hyvin, mutta ohjelmistoytimestä löytyneet ohjelmointivirheet hidastivat kehitystä jonkin verran. Opinnäytetyön kirjoitusprosessi viivästyi todella paljon, koska olin samaan aikaan täysipäiväisesti töissä, mutta prosessi saatiin lopulta päätökseen käyttämällä joka päivä hiukan työaikaa siihen.

Opinnäytetyön lopputuloksena simulaattori saatiin onnistuneesti yhdistettyä muuhun testausjärjestelmään ja ohjelmistopinon toiminnallinen testaus voitiin siirtää laitetestauksesta simulaatiotestaukseen. Tämä teki testaamisesta huomattavasti aiempaa nopeampaa ja monipuolisempaa, koska automaattiset testit voitiin käynnistää jokaisen koodimuutoksen jälkeen ja mahdolliset virheet löydettiin heti aikaisessa vaiheessa. Simulaattoriin pohjautuva testausjärjestelmä otettiin yrityksessä nopeasti käyttöön pääasialliseksi testaustyökaluksi ja 6LoWPAN-protokollalle on opinnäytetyön kirjoitushetkellä kirjoitettu jo yli 600 automaattista testiä.

LÄHTEET

1. Bormann, Carsten - Shelby, Zach 2009. 6LoWPAN: The Wireless Embedded Internet. West Sussex: Wiley.
2. Akyildiz, Ian Fuat - Vuran, Mehmet Can 2010. Wireless Sensor Networks. West Sussex: Wiley.
3. Olsson, Jonas 2014. 6LoWPAN demystified. Dallas: Texas Instruments. Saatavissa: http://www.ti.com/lit/wp/swry013/swry013.pdf. Hakupäivä 8.9.2015.
4. Directed Acyclic Graph. 2016. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Directed_acyclic_graph. Hakupäivä 14.3.2016.
5. RFC 6550 RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. 2012. The Internet Engineering Task Force (IETF). Saatavissa: http://tools.ietf.org/html/rfc6550. Hakupäivä 14.3.2016.
6. ARM mbed 6LoWPAN Stack Overview: Introduction. 2015. ARM. Saatavissa: https://github.com/ARMmbed/sal-stack-nanostack/blob/master/docs/02_N_arch.md. Hakupäivä 7.9.2015.
7. Connectivity: Mesh networking with 6LoWPAN. 2016. ARM. Saatavissa: https://www.mbed.com/en/technologies/connectivity/. Hakupäivä 13.5.2016.
8. NanoSimulator Documentation. 2014. ARM. Saatavissa: ARMin sisäinen dokumentaatio, vaatii käyttäjätunnuksen. Hakupäivä 7.9.2015.
9. CLI concept. 2016. ARM. Saatavissa: ARMin sisäinen dokumentaatio, vaatii käyttäjätunnuksen. Hakupäivä 22.2.2016.
10. Thread Technology. 2015. Thread Group. Saatavissa: http://threadgroup.org/Technology.aspx. Hakupäivä: 9.9.2015.